ROBOT ACTOR

PROJECT MEMORY

David Luna i Pérez

Tutor: Andrea Bonarini

Milano, February-July 2015

INDEX

1. INTRODUCTION AND OBJECTIVES

This project is based on the development of a robot actor. This concept is understood as a robot that takes part of a play or scene and has to perform a number of actions on that stage as if it was a normal actor.

The problem we find is that, as in all the plays, the elements of the scenario will never be in the same position at the same instant of the scene. This is the reason that the robot cannot be assigned a series of movements to do in specific moments when programming it and then press the play every time the scene starts because it would do the same movements independently of how the objects in the scene are located. The robot has to look at every moment of the play the movement that must be done, how and in which direction.

The robot used in this project is a three-wheeled omnidirectional platform called Triskar, created and developed by the university Politecnico di Milano used in many of its projects. The software that the robot was supposed to run is ROS but after some troubles with the ROS simulator Gazebo and a lack of availability of a robot with this software at the end of the term, the one that is finally used in the project is one controlled by an Arduino Uno board connected to a Serial, the device that receives the orders from the board and sets the speed of the motors.

To control the scene two fixed cameras are going to be used and they will be located in a specific place outside the scenario. Those cameras will allow us to know at any time the distribution of the objects in the scene, which means that we will know every moment where the robot and all the other objects and actors are located.

To carry out this location, we will need a program to make the image processing and a simulator to make sure that everything works fine before trying the code in the real environment.

The image processing program used in this project is the OpenCV, which is free, very versatile as it can be used in all the Operative Systems and with a lot of functions not only for image processing but also for receiving data from cameras or videos. With this software we will be able to search the objects in the scene and create a map of the scene calculating the estimated location of each one.

The simulator used in this project is V-REP. The controllers can be written in lot of different languages and even though the robot used in the project is not a commercial robot and then we are not able to have a good model for the V-REP, this simulator will be very useful to work on all the image software and the location and mapping code.

The programming language used on this project is Python. The reason to choose this language over others is that it is much more familiar to me and because Python has a structure that facilitates the reading to those users that are not familiarized with the code.

THE GOALS SET FOR THIS PROJECT ARE THESE ONES:

1. Elaborate a code able to create a map with all the objects on the stage and able to refresh it constantly. It will be necessary to see if the robot has moved in the correct direction and if the other obstacles have moved in that instant of time.
2. Elaborate a code that calculates the best path to reach a specific object or location on the stage avoiding all the other obstacles. It will be necessary to avoid that our robot collides with those objects.
3. Connect the robot and the computer so the robot can receive orders from the computer and make the desired move.

2. SIMULATOR

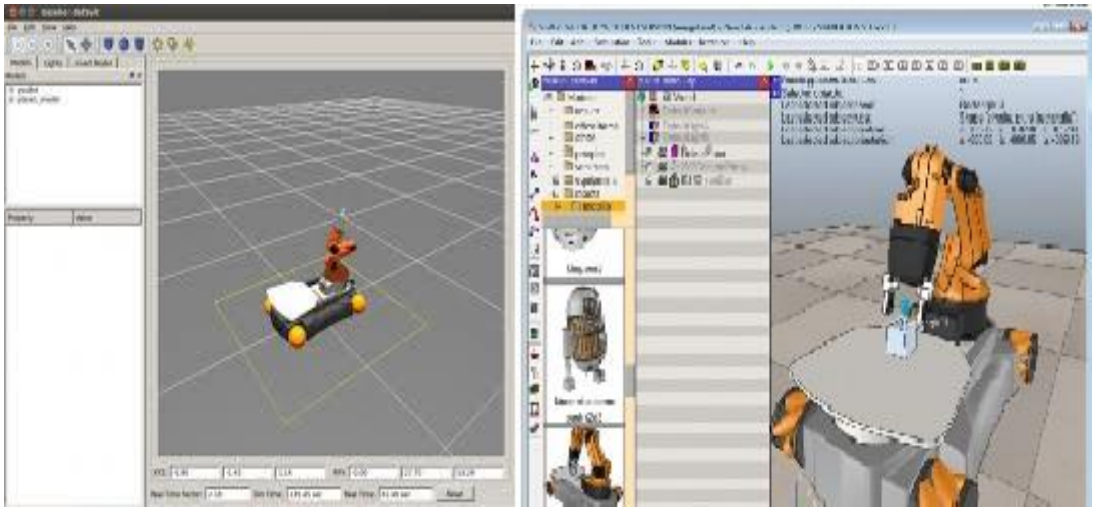
2.1 ELECTION OF THE SIMULATOR

The simulator used in this project is V-REP. V-REP is a robot simulators that enables the users to use a lot of different functions, more features than others and more elaborate APIs.

The robot simulator V-REP is based on a distributed control architecture. This means that each object or model can be individually controlled via a ROS node, a remote API client, an embedded script etc. Apart from that the controllers can be written in C/C++, Python, Java... This makes V-REP very versatile and ideal for multi-robot applications.

When comparing V-REP to Gazebo, the conclusion we can reach is that on one hand V-REP is a more intuitive and user-friendly simulator, and packs more features. On the other hand Gazebo is more integrated into ROS framework and is an open-source solution which means it allows a complete control over the simulator but it can be quite hard to use for those who does not have a strong background in robotics.

The main point to take V-REP simulator over Gazebo is that is much easier for both the modelling of the robot and the world that it is in Gazebo because there are lot of libraries with objects like tables, chairs, walls, doors… and is easy to place them and modify. Gazebo instead is only able to place geometrical figures like cubes or spheres and it is not very helpful to set a scene and even though there is a huge database on the Internet modifying the orientation or the size of the objects later is quite difficult.



Comparative image between Gazebo (left) and V-REP (right) interfaces

2.2 V-REP SIMULATOR

Once we have selected the V-REP as the simulator, we can start modelling the environment. As this is going to represent a scenario, the place designed will be a big area surrounded by walls except for the side where the virtual public and the cameras are going to see the scene and will include some daily objects like it can be a table or a couple of chairs.

Nevertheless, the modelling of the robot is much more complicated as its wheels have two movement possibilities, that controlled by the motor and that one free and perpendicular to the first one. As modelling this is quite hard and it was not a main goal in the project, instead of spending time in creating a model, one made by another student in a previous project was used.

Once the objects are set in the scene, we need to place the cameras too. In the V-REP interface we are given the option of placing cameras or placing vision sensors. The cameras can give images all the time while the sensor can only give them while the simulation is running but after looking in the API of V-REP, I came to the conclusion that that the object that will be used as a camera in this scene must be a vision sensor because even though the image quality is worse, there is a function that gets directly the image of the sensor while the image from the camera is much more difficult to get and heavier to process.

These sensors will be placed in both corners of the scenario where there is no wall and at a high of two meters. From this position we are able to see about 75% of the scenario in both cameras and the rest only from one, but considering that this 25% are the corners of the scenario we will not take them into account.

After running the scene some times and having problems with moving the robot, I realized that the movement code had some functions related to proximity sensors which will not be used in the real robot and that the model was of a four wheels robot instead of the three that the real robot will have. Due to this discrepancy between the real and the virtual robot and the complication of creating a robot model as said previously, the tutor of the project advised me to work with a robot already modelled in the V-REP libraries and with an easy code to make it move. After looking to all of them I chose a Kuka youbot, which is a robot that allows the user to make it move not only with a code but also with a controller that appears when pressing on the robot during the simulation.

To make sure that the localization code worked correctly instead of adding only one robot I decided to add two and see if both angles were taken correctly and that when calculating the position of both robots the angle were taken correctly.

We can open the final scene starting VREP by typing ./vrep.sh in the V-Rep folder and selecting the file finalscene.ttt that is in the Robot_actor zip. This scene is a place of 10x10 meters surrounded by walls except for the side where the public is supposed to see the scene. It includes as said before an extensible table, two chairs and two Kuka robots.

Although the simulator has not been very useful in his main function that is simulate the behavior of the robot when some orders are given to him, it became very important in the image treatment code because there were no shadows, reflexes, lights and people accidentally entering the scene or moving the cameras that should be fixed. That is the reason that the huge part of image coding was done in this virtual environment.

3. IMAGE PROCESSING SOFTWARE

The software used in the project to process the images is OpenCV. Even though python has its own image processing library called pythonvision, OpenCV offers the possibility to work much better with the image comparison and motion detection, which is at the end one of the main points that we will need in our code. This software has C++, C, Python and Java interfaces and supports Ubuntu Linux. It is the most popular and advanced code library for Computer Vision related applications nowadays.

In the python programs, this software works with numpy matrixes (arrays of arrays) but it can only support those whose type is uint8, which means that each element of the matrix represents a frame and is a number (from 0 to 255 in the greyscale images) or another array of three numbers [B G R], which indicates also from 0 to 255 the correspondent blue, green and red color in each frame.

The images got from the vision sensor in V-Rep are from another type as it is a numpy array which numbers can go from -256 to 255 so we need to change the type of the array before starting to work on it.

Those images loaded from a file or those frames got from a video or are directly loaded in the correct numpy type so when we are working with the cameras, we will not have to change these arrays type.

The image software will be used for search and locate all the objects that the cameras find in the scene. There are some ways to do it but I will compare the three easiest ones that this software allows us to use.

The first way would be by color finding. If we know the colors of all the actors and objects in our scene, we can make a color search program that looks for that color in both cameras and find the pixels where this color is. The main problem of this code is that in V-Rep it will work pretty fine because all the objects are always of the same color but when moving this code to a real scene,

we will find the problem that there are lights and shadows and the color we are looking for may be different from the one that we are seeing in the screen.

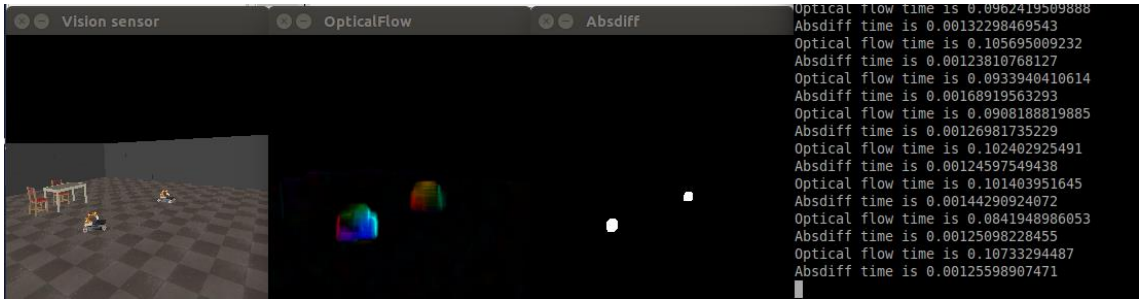
The second one would be by image feature extraction using a SIFT algorithm and matching those features from one image of every object and the scene. To make it clear, we would have the images of every actor or object in the scene and the code would look for a shape similar to this image in the scene. This method would be faster than the color detector because there will be no need to go over all the pixels of the image looking for the color but we will find two important problems in this method. The first problem is that the SIFT algorithm is art of the non-free OpenCV functions so to get it we will need to pay a license. The second and more important is that there will be two cameras with different perspectives of the scene and of course of the objects in it. We may find an object or actor which from one camera the view is exactly like the image we are looking for but from the other camera see a completely different face to that one we were looking for. We could also have images of some perspectives of all the objects but I think that there will be a lot of useless processing and it is not an optimal way to spend our resources.

There is another way for finding the objects in a scene that is by comparing the differences between two images taken in two different instants. All these functions are normally used to compare two consecutive images from a video or camera and see what has been moved between the two frames but in this case we are going to do something a bit different. The first image is one with the scene empty called background and the second image we are going to use is one of the actual scene in that frame. When comparing an image with the actors and objects and another with the scene empty, the difference between those two images will be precisely these objects we are looking for. There are some functions to compare two images in this way but the most used are:

Optical Flow. Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of these objects or the camera. There are two main optical flow methods in OpenCV. The

Lucas-Kanade method works with corner detection and detects where the corners detected in the first image are located in the second one and finds the vector movement of the object. The dense opticalFlow compares both images pixel by pixel and draws a stronger color as bigger is the difference between the two pixels.

Absolute difference. It is similar to the dense optical Flow because it compares both images pixel by pixel but this time draws the pixel in white if there is a considerable difference or black if the difference is not big enough. In the next page we can see the difference between both methods when getting the image of the absolute difference or the dense optical flow in a scene with two robots (finalscene.ttt)



Comparative between the dense optical flow and the absolute difference methods.

As it can be seen in the previous image, as the first method draws in all the pixels where a minimum difference is seen, the shape of both objects is seen much better than in the second method because only a little few of those pixels were detected as different enough to write a white pixel in this image. One think we may take into account is that the second method works much faster as it takes about eighty times less resources and time to calculate the images from both vision sensors in the scene. As the robots in V-Rep does not move fast and the first method offers us a better resolution of the shape we will take the first method to work in the V-Rep but if when working on the cameras the time rises a lot we may have to change to the second method.

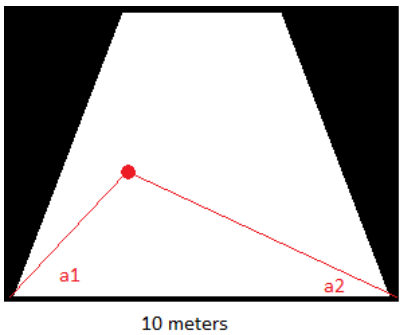
Once we get the image we need to look for the location of each object in it. In our case it means looking for the coordinates of the middle-bottom point. I chose this point because it will be the same if the object is ten centimeters or 2 meters high and will detect the central coordinate if the object is wider or thinner. To look for this point it is easier to look for the bottom-left and bottom-right corners and take the point in the average of these two points. There is the possibility to look for corners in the image but in this case as we can see a lot of squares with different colors, the code might find more corners than those who are really in the image so what I did is look for keypoints that are points in the edges of the objects and take the one more bottom-left and bottom-right of each object.

4. LOCALIZATION IN THE V-REP SCENE

Once we receive the images of the V-REP and process them with the OpenCV to get the optical flow and then calculate the middle-bottom point of every object in both images we have to transfer those coordinates to a specific location in the scenario.

As the scene is set with two immobile cameras fixed with a specific angle we can set, the easiest way to locate all the objects is by basic trigonometry. We know that the distance between both cameras is the width of the scenario that in V-REP for example is 10 meters. Even though in the real life this distance can change, it is easy to calculate. The other data we will need to place the objects is the angle seen in each camera between the object and the scenario edge.

The two options we have when calculating the place is by finding the distances between the object we are looking for and the camera or the angle between the object and the edge of the scenario. The first option is much worse because we need to have the dimensions of all the objects and we might find a view where a part of an object is unseen because it can be covered for another actor or object and then seem much smaller than what it really is. So the option taken to make these calculus is looking for the angles between the cameras and the edge of the scenario. As seen below, when we get the value of those angles we will be able to place the red point (which represents the object to locate). The black zone is that space where the object is not seen by one of the cameras.

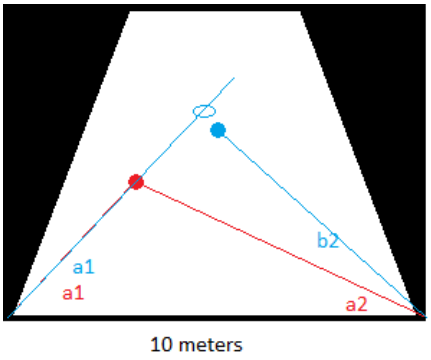


Scheme of the normal localization process

The vision sensor can provide us the angles of all the objects in each image because it can be considered as a point and it is possible to graduate the opening range of the angle. It means that every pixel in the horizontal and vertical image represents a determinate angle that is exactly the opening range divided with the number of pixels in the image (70/128 degrees each pixel in this case).

Once getting to this point it must be set clear that the result of the localization will be good if both angles are quite big but if they get small we might find some problems as there is a sinus division for getting the distances between each camera and the object and when reducing a lot one of the angles, the sinus tends to 0 and the division can make the number searched much bigger than that expected. This is the reason that for angles smaller to 10 degrees the result started to differ much more from the real one.

Another problem we might find is that one camera detect two objects while the other does only detect one. It is possible if in one camera the first object seen interferes in the vision of the second object. The problem can be solved considering that the angle in that camera is the same for the two objects even though it can be different as seen in the image below



Scheme of the localization process when found only for one camera

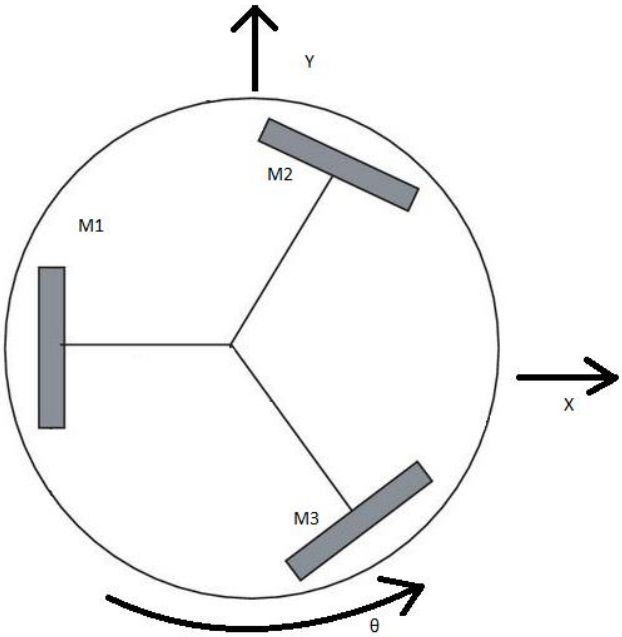
Of course the difficulty grows as it grows the number of objects to locate in the scene. As the scene is quite extended, we could rise the cameras or introduce some others in other corners of the scene to make a better triangulation, but as we are finally working in a real scene that might not accomplish these characteristics we will not introduce more changes to the code.

If we run the V-REP scene and execute the localization code while moving the object we will see that the theoretical place calculated with this code and the real place of the robot in the V-REP has a difference of not more than 0.1 meters (which considering that the scenario is 10 meters width is a very small error) except for that zone, as said before, too near to the edge of the scene.

5 THE TRISKAR

5.1 THE OMNIDIRECTIONAL PLATFORM

The triskar robot is an omnidirectional platform created and developed by the university Politecnico di Milano. This robot has three Omni wheels that are wheels with small discs around the external circumference placed perpendicularly to the wheel rotation direction. Thanks to this, the wheels allow free lateral movement in the perpendicular direction of the wheel plane but can also move in the direction parallel to the wheel plane thanks to the motor force. As the robot has three motored wheels and three movement possibilities (x direction, y direction and rotation), we are talking about a holonomic robot, which on one hand allows grater maneuverability and efficiency in the movement but on the other hand it adds some complexity to it.



Scheme of the omnidirectional robot and its movement possibilities

The advantage of having an holonomic robot is that any movement we want our robot to make, there will be only one combination of the motors speeds that allows the robot to do them and this combination will be the linear

sum of what is needed to do each of the single movement (x,y and θ). The process for receiving the velocities of the motors given the speeds wanted is the inverse kinematics and it is just calculating the inverse of the matrix. This result will be what we are going to apply to the Serial of the robot so that we only have to send the desired speeds to make the robot move without operating the matrix.

The kinematic matrix for the three wheeled robot is the next one:

$$\begin{vmatrix} Vx \\ Vy \\ \omega \end{vmatrix} = \begin{vmatrix} 0 & \cos(30) & -\cos(30) \\ 1 & -\sin(30) & \sin(30) \\ d & d & d \end{vmatrix} \cdot \begin{vmatrix} V1 \\ V2 \\ V3 \end{vmatrix}$$

5.2 THE CONTROL

The Serial is a device connected to a pin of the Arduino Uno. This device gets the information from the Arduino Uno board and sets the speed of the motors as required by the user.

Arduino is an open source prototyping platform. It was developed as an alternative to the expensive boards existing before its appearance and has spread a lot in the field of the robotics

The Arduino Uno is a microcontroller with 14 pins to input or output information and a USB connection that is the way the robot is programmed. A microcontroller is an integrated circuit that combines several functional blocks. The main parts that form the microcontroller Unit are the CPU, the memory, and the peripherals. This is also, a device quite cheap, versatile and programmable so it is possible to perform functions and provide very complex systems.

The programming of the microcontroller present in these devices is done through a simple and intuitive own language, Arduino, that favors fast learning. This is based on the high-level language Processing and C, which supports many functions. To load that code we will need the Arduino program in our computer

and once the code is finished, we press the compile button and then the code will be saved, compiled and uploaded to the Arduino Uno.

This board can be linked to the computer using the USB cable, or adding a receptor of Bluetooth or Wi-Fi. The problem found were that the optimal Wi-Fi connector for the board, called xBee were none available in the university and the version needed were not sold in the shops of Milano. On the other hand, although the Bluetooth connector was also hard to find, there was the possibility to buy one in a shop outside the city but after mounting it on the Arduino Uno board I tried to link both devices but it was not possible. Later I found out that the computer used is not able to connect via Bluetooth.

If we had one of these possibilities, we could use the python library pySerial that allows the user to give orders directly to the Serial through the receptor and the Arduino Uno board. We could create in the Arduino code a function which could be called with the speeds desired and that sends the serial the speeds of the motors wanted.

As in this project we are not given this possibility, the only way out then is to use the USB cable. This has an obvious problem in a scene and is that the cable can entangle with itself or with other objects in the scene, even become an obstacle for the robot. Apart from that, we may need a very long cable to make sure that wherever the robot is in the scene, the cable is long enough to reach that position and send the information. There is another problem and it is that if the cable has a different color from the background we may have problems with the location code so that cable should be of a color very similar to the background.

For all that reasons, I considered that the cable is neither a good way to transmit the information to the robot if it is placed in the scene and that is the reason why, despite what at first wanted, the project does not incorporate the live transmission of the information.

Although in the introduction is said that we cannot assign to a robot actor a series of moves to do in specific moments but we have to look at every moment of the play the movement that must be done, how and in which direction, that is the only alternative found to the live transmission of the information. We lose this way the possibility to control moving objects or other actor's position but if our scene has no need of this type of actors, the loos will not be important as one thing we can actually control is the theoretical position of the robot in each instant of time.

The advantage of introducing the code to the robot at the beginning of each play to a robot with the same code in each scene is that we can set the position of all the objects in the scene however we want. It means that every time we set the objects, some can be added, deleted and changed its position. Later we cannot control if they move or if more objects are added to the scene.

5.3 THE MECHANIC ELEMENTS

5.3.1. THE WHEELS

The wheels are the most peculiar part of the robot. As they are omnidirectional, they must be able to force the robot to move in the direction of its plane in the speed of the motor but they must also let a free speed in its perpendicular direction. To get that possibilities we are not having a conventional wheel but one with cylinders tangential to the wheel like in the following picture.

Picture of an omnidirectional wheel

This configuration allows the movement expressed above. Thanks to it, this wheel can not only move in one direction of the floor but in all of them. To calculate the speed needed in the motor to make the wheel move in the desired direction, we have to decompose that speed in the one perpendicular to the plane of the wheel and the other parallel to that plane. If for example we want the wheel to move in 60 degrees, the speed of this wheel must be half of the speed we want the robot to have.

5.3.2 THE ELECTRICAL CIRCUIT

As in all the electronic and devices with electrical motors, we are going to need a electrical circuit for the elements to get their power.

The most important thing in the electrical circuit is the generator or battery, in this case it will be a battery with a maximum charge of 5000 mAh and a nominal tension of 11 V that will be connected to different cables that give the power to the Arduino board, the serial and the motors.

Although the Arduino has an input voltage of 5V, the board can regulate the voltage entrance and there will no need of a tension reducer for it. Actually if we look at the table of characteristics of the Arduino Uno we will see that the recommended input voltage is from 7 to 12 V.

The other devices work perfectly at 11V but can start failing and giving bad results as the battery power starts to get down.

There is also a low voltage connection between the 11^{th} and 12^{th} pin of the Arduino and the Serial by two cables. These cables are the ones that send the information to the Serial for this device to actuate on the motors and make them move in the wished speed. In that case, the

power does not come directly from the battery but it is first regulated by the Arduino board.

To control when the device starts to run or when it has to stop there are also two switches at the edge of the robot. The first one is linked only to the Arduino board and the second is linked to the Serial and the motors. When turning on the switches, I recommend to turn first on the one of the motors because when the Arduino gets the power it starts automatically the code and if from the first millisecond the robot has to start to move, we can lose the movements to be made until we connect the motors to the power.

6. WORKING IN THE REAL SCENE

6.1 PREPARING THE SCENARIO

Once the virtual work is done and we know how our robot works, we can start working in a real scenario to represent the play. In our case this place will be a zone inside the AirLab of the Politecnico di Milano which is the room where all the students of the Robotics department made their projects. The area that will become the scenario is more or less a square of 3.5 meters for 3.5 meters. Even though the space we will have is not the same as the one in the V-REP the code can be perfectly adapted.

The devices used to capture the images of the scenario are two cameras Logitech C270. As said previously, these cameras are supposed to be fixed, immobile and placed in a high position. For this I used two metal structures of about 2 meters high and placed the cameras on the top of them. With these structures, the cameras got all those characteristics for the positioning but it was not possible to get a definitive place for them as when finishing to work every day I had to tidy up my zone of the lab and then place the structures again the following day.

6.2 THE MAPPING CODE

As the structure is not placed every time in the same place we have two options. The first one is to adapt our previous code so we can introduce at the start of the program the characteristics of the stage and the distribution of the cameras and its orientation. The problems we may find are that the X direction of the camera may not be placed entirely horizontal and that to get a good result we have to take accurate measures of distances and angles between the metal structures and it is quite complicated to not make any mistake in this process.

The second option is by placing the cameras in any distribution and then calibrate them, which means that we can place an object in some different coordinates in the stage and make a file with the coordinates on the stage and the coordinates of the middle-bottom pixel of each camera. This way the scene can be set anywhere and the user does not have to make any calculus, all that work is for the computer.

Once we finish the file with some coordinates there are two options to calculate the position of the objects that the camera may find when running the location code. The easiest is to make a linear regression for both x and y coordinates with the variables C0x, C0y, C1x, C1y with all the points taken. Those variables are the coordinates of the pixel that contains the middle-bottom point of the object in each image. Python has a statistics library that does automatically a regression given the parameters and the dependent variables. We can also get the R square value and we can see after doing some regressions that this value goes from 0.84 to 0.98 depending on the distribution of the points, which is a really good number.

There is also the option to, when given the pixel coordinates of the object look in the file for the three or four nearest and then make an interpolation of the coordinates to reach the coordinates x,y where the object tracked is supposed to be. Nevertheless, this way only one image is used and every loop we have to read the file as times as objects are in the image and as the file can be quite big, it can slow the program too much.

So there are two programs that need to be done. The first one that is the program that allows us to calibrate the cameras and writes the file used for the posterior tracking and the tracking program that uses the information written in the file made in the previous program and when an object is detected in both cameras it can calculate the x and y coordinates.

Once these programs are done correctly, when trying to run them we can notice that the image will be refreshed very slowly. The reason that the program

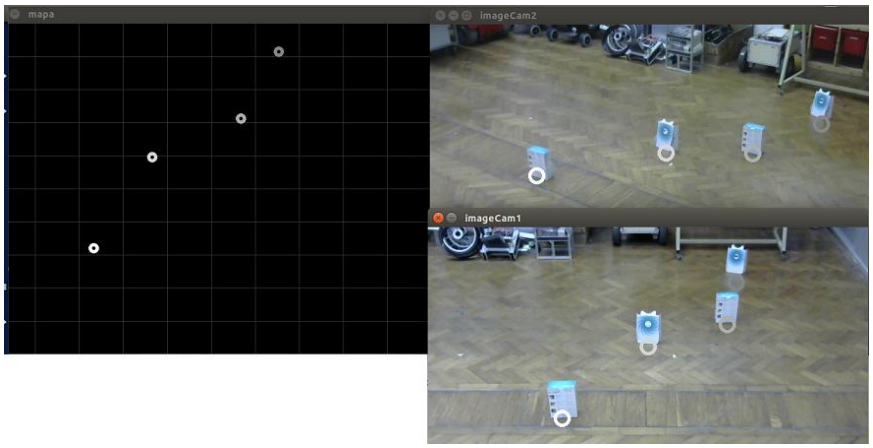
works so slow is that the images last a lot of time to be processed. The size of the images given by the real cameras are of 640x480 pixels compared to the ones given by the vision sensors in V-REP, the images from the camera are 19 times bigger than the others and lasts 19 times more time to calculate them, more or less 2 seconds for each loop. I consider that this time for each loop is too much because the scene can change a lot in 2 seconds and it is necessary to reduce that time. As said in the OpenCV section, there is the option to use the AbsDiff method that, even though we may lose the shape of the object, it works much faster and we can find equally the middle-bottom pixel of the object because it will not differ from the real one. Like this we can refresh the image a bit more than times per second and that is the reason that we will see the scene much more fluid that with the other image processing method.

So the steps to find the theoretical coordinates are these ones. First of all we take an image from both cameras and look for the objects in it. As said in a previous section, we compare the image of the background with the image from the camera and the difference are the objects and once we have all the objects detected we look for the middle-bottom pixel of each object. The next step we need to take is that we need to know the correspondence between these points from one and the other image. To make it clear, there will be a list with the points found in the first image and another list with the points found in the second and we have to make sure that those lists are sorted so that each point has its position in the images in the same position on the list. If the lists are not sorted like that, we will use in the regression two variables well and two wrong so the position will differ from the real one.

The code created to solve that problem works in the following way. First of all we load the program used for the regression and we create a list where every element is another list containing the C0 and C1 points. Then we get the list of the middle-bottom pixels of the first image and we look for the nearest C0 point. Once it is done we take the C1 that is the second element of the list where the C0 found is and then look for the nearest middle-bottom point in the second

image list. This is the reason that if when running the program we have a problem with the correspondence we need to take more points for the C0 and C1 list or to separate more the objects in the scene.

The next step taken was writing a code that given the x,y positions of the objects in the scenario and its dimension, creates a map as it is easier for the user to control visually the position of the objects and see if the code works fine and if the error. In the following picture we can see the images got from both cameras, the middle-bottom pixel of each object in the scenario, that the color in each object is the same in both images (it means that the correspondence is done right) and the position that the objects have in the map. (Notice that even though the scenario is 3.5 for 3.5 meters, I divided it in parts of 35 centimeters so it can have 10 units width and 10 units high)



Vision of both cameras and the mapping of the objects found in them

6.3 THE MOVEMENT CODE

In this project we will only work in the basic actions such as reaching a coordinate or an object but as in that way to that point we can find some obstacles, our code must be able to write a route to avoid them.

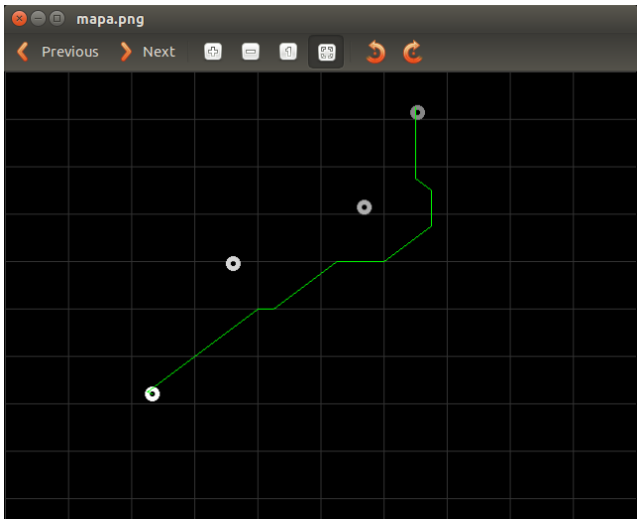
Once the virtual map of all the objects that compose the scene is made, we can start thinking in how the code to move the robot will be. In the following

paragraphs the concept step is used. I call a step to that part of the track between two points and the sum of all the steps makes the route or path. There are two ways to make this route successfully:

Free route. In this code the robot does the steps straight to the object wanted. If it finds an obstacle in his way to that object, the robot will change the direction of his route getting far from the obstacle found until it finds a good point to finish the step. This code is much more natural because the robot goes straight to the place desired but I found a problem when trying to pass between two obstacles very near one with the other.

Tiles route. In this code, the scenario has limited possible positions for the robot instead of having infinite possible positions. The map is divided into smaller pieces called tiles (in this code 40 tiles in each direction x and y). We have to know for each tile if they are visitable, if they are too near to an object or obstacle (except the robot and the object we are going to visit if doing so) they will not be visitable. The route is made then by searching the smallest tile track to the end.

After generating both codes and trying them in some cases, I reach the conclusion that despite the first is more natural for the reason exposed above, there was a problem that I could not solve and it is that when there were two obstacles one near each other the final point found when avoiding the first object area was inside the second object area and vice versa and for that reason the code started an infinite loop because it just jumped from one obstacle area to the other. I tried to correct this but in this case the solution was worse than the problem and I decided to take the second code to make the movement. Below we can see an example of the path from one object to the other avoiding those in the way.



Path calculated by the computer to connect the robot with an object

We then only have to concatenate the goals to reach by the robot to create the scene. In the section 9.4 it is explained how it is done. Once we get all the points to visit we need to translate it to orders that the Arduino gives to the Serial for the movement of the wheels.

The structure of the Arduino file is the following:

Variable declaration, where the needed libraries are called and the variables needed in the code are given a value

Void setup(), that is the function called only once and the first of all. This function initializes the serial and starts to count the time.

Void loop(), that is the function that goes repeating on and on until the Arduino is switched of and in this case this function only calls another that is scan()

Void scan(), this is the function called in the loop and here it is where all the code will be placed. To run the device we give some speeds to the Serial during a determined time. The structure of this code is easy, first there is the condition that in our case will be related with the time and then it is the order.

A simple code would be this one:

```
if(millis()-tStart<2000)  {
serialRuote.println("r 0 0 0 ");
}
else if(millis()-tStart<6000)  {
serialRuote.println("r 0.4 0. 0 ");
}
else  {
serialRuote.println("r 0 0 0 ");
}
```

If we just write the points visited on a list and then calculate the direction between each point and the distance and given an arbitrary speed, we can calculate easily the time and the speeds needed for the motors to follow the path and knowing them it is possible to create a code that writes in a .txt file with the orders that must be given to the robot to follow the route.

7. INSTALLATION GUIDE

This section is a basic guide for the installation process. Before starting to work in the programming and the scene the computer must be prepared and for it, some programs must be downloaded and we have to make sure that they run correctly.

The first thing needed is a computer supporting the Operative System Linux. If the computer is already able to run this OS you can proceed to the step 2, otherwise start from the first step.

7.1 INSTALLING LINUX UBUNTU IN YOUR COMPUTER

There are three main Operative Systems in the market nowadays, if the computer does not have Ubuntu, it surely will have Mac OS if the computer is from Apple or Windows OS otherwise.

It must be decided whether we want to have Linux as the Operative System of all the computer, of a partition of the disk or as a guest Operative System in a Virtual Machine. We could run it from an external memory as well but it usually works slower and it can have some problems. So the three options I will consider are:

LINUX AS THE MAIN OPERATIVE SYSTEM OF THE COMPUTER

This option is only for those computers that are going to work from now on in Linux as the main Operative System. For example if the computer is going to be used only for our project. On one hand, in this way the computer is going to use all his potential in Linux and everything will work smoother and faster. On

the other hand it is going to lose all the programs and files it already has so we have to make sure that we save somewhere everything we want to keep.

ONCE IT IS DONE, WE CAN START THE INSTALLATION PROCESS:

1. Download the Ubuntu CD with the latest free version in the page http://www.ubuntu.com/download/desktop . The file downloaded is an ISO file and we can place it in an external DVD or CD by clicking with the right button and pressing open with > Windows Disc Image Burner> Burn if we have Windows or Applications>Utilities>Disk Utilities and burn if we have a Mac. In my case was the version 14.04.
2. Restart the computer and boot the CD with the Ubuntu.
3. Ubuntu will tell you if you want to delete the previous Operative System or just install it alongside it. In this case we will click the delete previous OS
4. Follow the steps indicating the language, the zone you live, the keyboard distribution… and then press install.
5. The installation will take some minutes, once it has finish, just restart the computer and it will run the Linux OS.

LINUX AS ANOTHER OPERATIVE SYSTEM OF THE COMPUTER

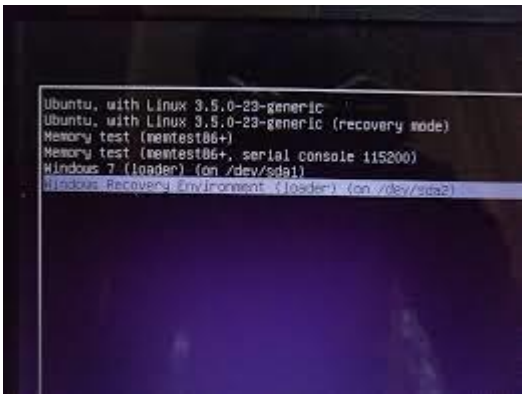
This option is for those computers that are going to work a lot on Linux or that need a lot of memory and resources when this OS is on. When starting the computer we will be able to choose between the previous OS we had in the computer and Linux. On one hand, we will have two OS in the computer with a lot of resources and we are not losing any programs or files. On the other hand, we can only run one OS at a time and they cannot be linked in any way.

THE INSTALLATION PROCESS IS QUITE SIMILAR TO THE ONE EXPLAINED PREVIOUSLY:

1. Download the Ubuntu CD with the latest free version in the page http://www.ubuntu.com/download/desktop . The file downloaded is an ISO

file and we can place it in an external DVD or CD by clicking with the right button and pressing open with > Windows Disc Image Burner> Burn if we have Windows or Applications>Utilities>Disk Utilities and burn if we have a Mac. In my case was the version 14.04.

2. Restart the computer and boot the CD with the Ubuntu.
3. Ubuntu will tell you if you want to delete the previous Operative System or just install it alongside it. In this case we will click the install with other.
4. Follow the steps indicating the language, the zone you live, the keyboard distribution… and then press install.
5. After the installation is done, restart the computer and there will be a screen with the option of opening the Linux OS or the previous we had.



Screen where we can choose which OS launch in our computer

LINUX AS THE OPERATIVE SYSTEM IN A VIRTUAL MACHINE

This option is for those computers that need to use Ubuntu but not permanently or with a lot of capacity. This option will keep the main OS in the computer. On one hand, we are not losing any programs, files or memory space and we can link the host OS with the guest. On the other hand in this way the calculus can become very slow if we do not prepare the VM properly. This is the option I took for my project.

THE STEPS TO GET UBUNTU IN THE VM ARE:

1. Download the Ubuntu CD with the latest version in the page http://www.ubuntu.com/download/desktop. In my case it was the version 14.04. It is free.
2. Download and install the programme for the VM. In my case I used Virtualbox and the programme for installing can be downloaded at www.virtualbox.org
3. Run the VirtualBox and press New
4. Name the machine like Ubuntu 14.04 with the OS Linux and version Ubuntu.
5. Select how many RAM and physic memory you want the Linux to use. My recommendation is giving to the RAM half of the computer RAM capacity and between 10 and 15 Gigs in the physic memory. If later more capacity is needed there is the possibility of changing that numbers later.
6. Select as the starting disc the file downloaded from the Ubuntu page.
7. Once the setup of the Virtual Machine is finished, press Run.
8. Follow the steps indicating the language, the zone you live, the keyboard distribution… and then press install.
9. The installation will take some minutes, once it has finish, just restart the Virtual Machine and it will run the Linux OS.

View of the Virtual machine on another Operative System

10. The last thing we have to do is prepare the Virtual Machine for the external USB devices, the cameras. Once we have the cameras plugged in the USB and of course all the drivers installed in the main OS, we have to open the main Virtual Machine window, which is the one where we choose the Operative System we want to run, select the Linux operative system and press Configuration.

    Then we go to the USB page and make sure that there is a tick in enables USB controllers and enable USB controllers 2.0. Once it is done we press the button at the left with the USB and a green symbol of + and we add the two cameras. Once it is done we have to make sure too that there is a tick in the cameras name and then press accept and run the virtual machine.

7.2 INSTALLING OPENCV

OpenCV (open source computer vision) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Ubuntu Linux. It is the most popular and advanced code library for Computer Vision related applications today. It is the main tool on our project as the image subtraction from the cameras and the processing is basically OpenCV based.

FOR INSTALLING THIS TO THE COMPUTER WE HAVE TO FOLLOW THIS STEPS:

1. Make sure that our system is updated and upgraded. We will open a terminal and write:

    sudo apt-get update

    sudo apt-get upgrade

2. Then we will run the installer typing in the terminal

    sudo apt-get install libopencv-dev python-opencv (or libcv-dev if the package is not found)

If we get an error typing the previous command we will have to install the program manually.

3. Install the dependences of the openCV program. This dependences are for example the build tools, image and video, libraries, Java, python… Just type the next command:

   sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk libtbb-dev libeigen2-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev qt5-default libvtk6-dev

4. Go to the OpenCV official website (opencv.org) and download the latest version for Linux. Then decompress this file.
5. Open a terminal, go to the OpenCV folder and run the commands

   -mkdir build

   -cd build

   -cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/..

In the previous command change the two points (..) with the path to openCV source directory

   -make

   -sudo make install

   This will start the installation process.

To check that everything works properly, we can open a terminal, executing the command python and trying to import cv2. If there is no error it means that everything worked fine.

7.3 INSTALLING V-REP

V-REP is one of the robot simulators that enable the users to use more different functions, features and more elaborate APIs.

The robot simulator V-REP is based on a distributed control architecture. This means that each object or model can be individually controlled via a ROS node, a remote API client, an embedded script etc.

Apart from that the controllers can be written in C/C++, Python, Java... This makes V-REP very versatile and ideal for multi-robot applications.

TO DOWNLOAD THIS PROGRAM WE HAVE TO FOLLOW TWO SIMPLE STEPS:

1. Go to the webpage http://www.coppeliarobotics.com/downloads.html and download the V-Rep pro edu file in the OS we are using (either 32 or 64 bits)
2. Unzip the file

To check that everything was installed properly, we can go to the directory where we have the folder of V-Rep and typing ./vrep.sh and if the program is executed, the installation has been correct.

7.4 PREPARING THE FOLDER FOR THE CODE.

This is the simplest step of all. Just to take the file Robot_actor.zip and unzip it. In this folder we have apart from all the python code a black image that is necessary for doing some processes with images and a scene of V-Rep that is the file that finish in .ttt that is the scene that I have worked on during this project.

We can move all this code to any folder in our Linux OS, if the installation is done properly, it will work.

7.5 LAST DOWNLOADS

Once all this main programs are downloaded, there are some other things that are recommended or different for every user:

1. The camera drivers. Depending on the camera we are going to use, we need one driver or another. Normally, when the camera is plugged in on the computer the drivers are installed directly but sometimes it does not happen and these drivers have to be installed from the webpage of the company.
2. Arduino. In this project, our robot is going to be controlled by an Arduino Uno and all the programming related to the robot is based in this particular robot. Arduino program can be directly downloaded from the webpage www.arduino.cc. If the robot uses for example ROS to move we should follow the steps for installing it and all the programming related to it should be changed. In the Arduino program we have to choose the board we are going to program, in this case the Arduino Uno and the COM portal where we are plugging the board to the computer. If the robot is plugged there will be only one number available but if it is not or there are many options we can see the number in Control Panel → Hardware and sound → Device administrator.
3. Geany. Geany is a text editor that allows the user to execute the program doing a simple click. This is not an obligatory program but it is very useful when writing the code but it is also quite helpful when we have to run some programs on a row like it will happen in our project. To install it just go to the Linux program searcher on the top-left corner, look for Geany and install it.

8. USER GUIDE

This is a basic guide for all the users of the Robot actor programs. In the following pages the way to setup the scene are explained step by step and some images that will help to see better what it is done.

FIRST OF ALL WE HAVE TO MAKE SURE TO HAVE ALL THE NEXT OBJECTS:

- A computer with all the programs explained in the installation guide installed
- Two cameras, better if they are of the same brand and model, and their drivers installed on the computer
- The omnidirectional robot with the Arduino Uno
- Some other actors for the scene.

Once we are sure we have everything we can start setting up the scene:

8.1 PREPARING THE CAMERAS

In this project we will work with two fixed cameras that will control everything that happens in the scene. To begin with we need to get a place where to install the cameras and make sure that they will not move. This place must be high and all the scenario should be seen from both cameras.

It is important to accomplish two requirements in this scene. The first and more important is that all the scenario must be seen in both cameras. To make sure that it happens in this way we can walk through the limits of the scenario and see if both cameras can see us. The second is that the biggest object or actor in the scene should not be bigger than 5-10% of the image. If it is bigger we may find some problems in the later processing. If there are objects such as tables or columns, we will just consider it as a part of the scenario. Otherwise if there is an object bigger than this 5-10% we can use a Len to expand the view and make the

object smaller or just install the cameras somewhere further or higher with another angle.

To make sure that the cameras are well positioned, the best way is to see in the computer the both images that they capture, so we can execute the file seeCameras.py in Geany by pressing the button run or open a terminal, go to the Robot_actor folder and write the command “python seeCameras.py” (When we want to stop the program we can close the terminal or press ctrl+c). We will see something similar to:

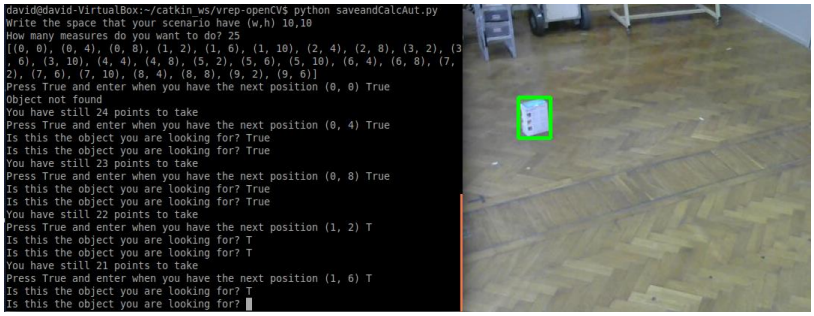
View of the two cameras that will help us to place them.

Once we see the entire scenario from both cameras, we need to clean it of any objects (excluding those who are part of the scenario) and then run the program saveBGcamera.py that will save the background image of the left and right camera.

8.2 CALIBRATING THE CAMERAS

Once we have prepared the cameras for the scene, we need to calibrate it, which means that when detect an object is detected in both cameras, we have to get the position of this object in the scenario. To do this, first we have to take some measures and there are two ways to do it:

Getting the points automatically: In this option, we give the high and width of the scenario, the number of points we want to take and a list of points is created with them distributed homogeneously. After that we proceed to put an object in all these points and a file is created with all the x,y coordinates and the coordinates of both cameras. The program we have to run is saveandCalcAut.py



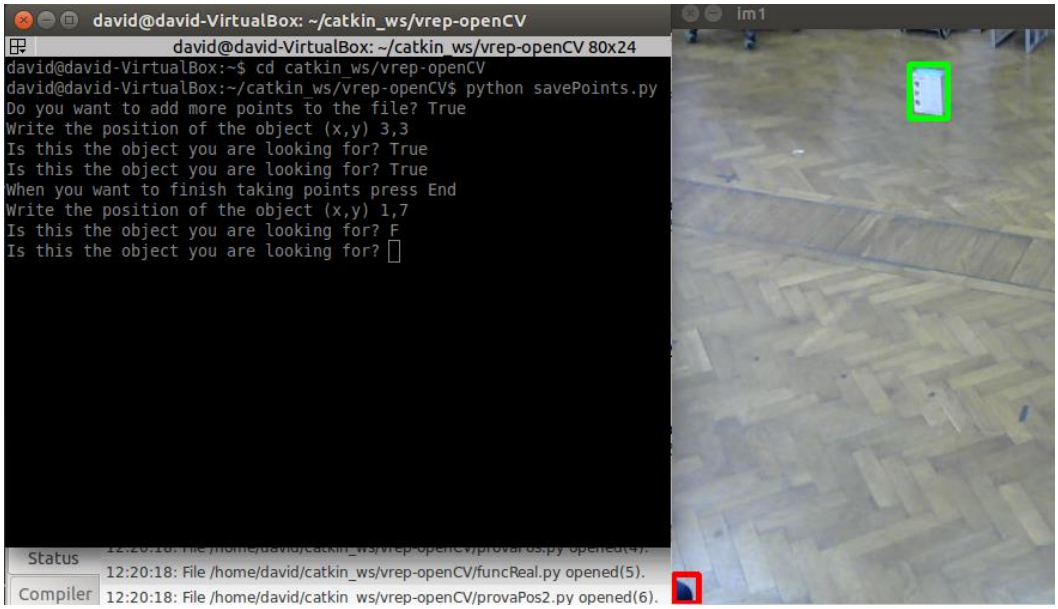
View of the screen when running the program saveandCalcAut.py

Introducing the points for ourselves: In a lot of scenes, the big part of the action is situated in a small zone of the scenario and not in all of it, when we can foresee that there will be a lot of actors in a specific place, we should take a lot of measures in that zone. In this option we write each point we want to calculate so we can group most of these points where the scene is going to take place. To run this program we have to write in the terminal “python savePoints.py”

In both cases when we are looking for a point, two images will be created and there will be a square for all the objects detected in the camera. It is useful in case there is more than an object in a camera, for example if something unexpected has appeared in the scene. If one camera does not detect any object,

a warning saying “Object not found” will appear and this point will not be written and we will proceed to take the next. To finish taking points we have to introduce ‘End’ when asked what the next point we are going to take is.

As there are a lot of points to be taken for the code to work properly, I recommend at least 30, and given the possibility to take our own points, I think that the best strategy to calibrate the cameras for our scene is to run the first program and ask for 25-30 points and then run the second and take important points like those where an immobile object will be placed or a zone that we may have a lot of objects.



View of the screen when running the program savePoints.py

8.3 PREPARE THE SCENE

Here we are going to set up the scene as it must be at the beginning. Obviously in the scene we will have some objects that are going to be motionless and some actors that will move.

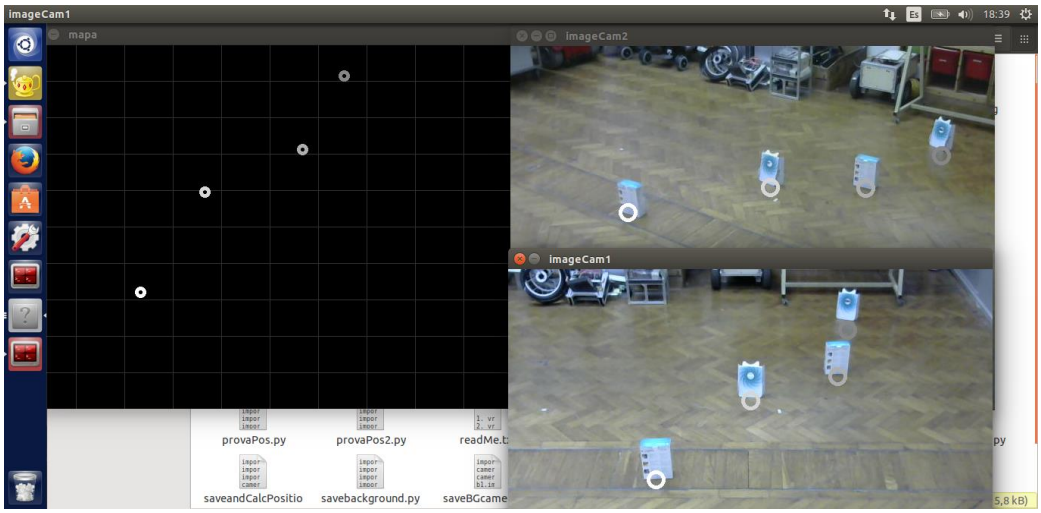
IT IS IMPORTANT THAT ALL THESE OBJECTS AND ACTORS MEET CERTAIN REQUIREMENTS.

- All the objects and actors must be opaque, without holes and with nothing that can be confused with the background of the scene. It is recommended that they have only one color but it is not obligatory.
- As said in a previous step, the biggest object should not fill more than a 5-10% of the image in the camera.

Once all the objects and actors are settled, we can run the program with the command “python searchPositions.py”. We will receive three images; two from both cameras and one that is the map of the scene. We can see in a glimpse if in both cameras all the objects are detected and the position is correct in the map.

If there are more objects detected that those in the scene it is possible that one of the cameras has moved. The other option is to find an image with an object with two circles, which means that this object has been detected like if there were two objects maybe because the color is too similar to the background or maybe because it has a hole.

Otherwise if we see a circle somewhere in the floor it can be a problem that a shadow or a reflex of an object has been detected as an object.



Result of the previous command with a distribution of objects in the scene

It is also the possibility that the number of circles are right but discover that the map is not correct. The main reason for that is that two objects have exchanged their positions in one of the images and the result of the calculus to find the position in the map is wrong. We can see in the images that they have the colors changed. To solve this problem we have two options: The first one is to separate a bit the objects if the scene allows us to do so and the second one and better is to get some more points near that zone using the command "python savePoints.py" as explained in the previous step.

If there are less circles than objects it can be for two reasons. The first one is that there are two objects too near that they are detected like if it was one or that there is simply an object too small or too similar to the background that it is not detected. If it is the first option we need to separate a bit those objects and if it is the second one, we should change the color or make the object bigger.

If we are not able to solve the problems, we can run the program absDiff.py to get the absolute difference images. These images are totally black except for those points that in the background and in the image are different, which there is a white pixel. The images can be very helpful to solve problems and to understand a bit how does the code work.

8.4 STARTING THE SCENE

Once the scene has been settled, we will work with the program startScene.py.

If we had the option to connect both devices (Arduino and the computer) we could calculate in every loop the path the robot must follow but, as we do not have the option to connect the robot with the PC wireless, the only alternative we have is to prepare all the scene before it starts, giving to the robot all the actions we want him to do and then load these instructions to the robot

and turn it on. There are two main handicaps. The first is that we can only work with immobile objects and actors and the second is that we cannot know if our robot is doing the path correctly.

Anyway this program is ready to make the robot move to any coordinates (x,y) of the map and to the position of any other object/actor. **There are two functions that calculates the path to a coordinate or to another object that are:**

mapa,posList=ArdCode.gotoObj(position,actor,actors,mapa,rangeMap)

position: Is the place where we start, the first function will have posR and the others posList[-1]

actor: Is the name of actor or object where we are going.
All the other variables must not be changed

mapa,posList=ArdCode.gotoXY(pos0,pos1,actors,mapa,rangeMap)

pos0: Is the position where we start, in the first function we will write posR and the others posList[-1]

pos1: The coordinates of the place we are going to.

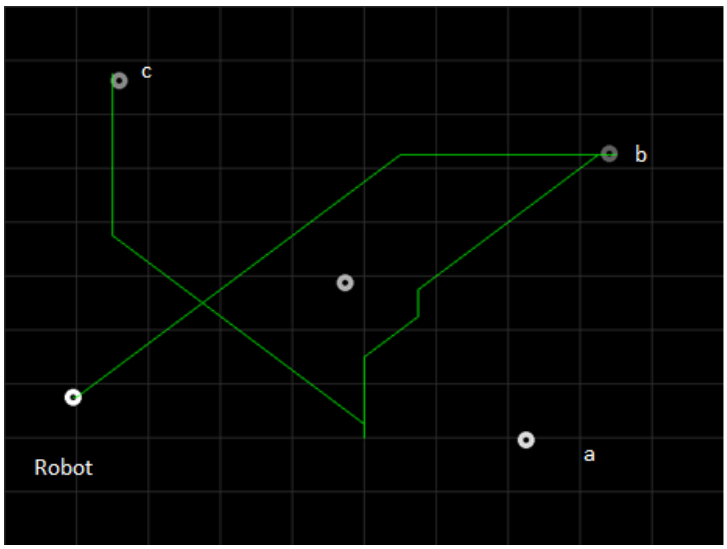
All the other variables must not be changed.

In order to have a list with all the positions we are going to be for the Arduino file, we have to save all the positions where we are going in every movement. To make this we will sum all the posList like in the following example:

```
mapa,posList=ArdCode.gotoObj(posR,'b',actors,mapa,rangeMap)
mapa,posList0=ArdCode.gotoXY(posList[-1],(5,2),actors,mapa,rangeMap)
posList=posList+posList0
mapa,posList0=ArdCode.gotoObj(posList[-1],'c',actors,mapa,rangeMap)
posList=posList+posList0
```

In this example we are going from the place where the robot is to the object called b, then to the coordinates (5,2) and to finish to the object called c. The code made like the explanation below is written in the file startScene.py

from the line 73 on. We can delete this five lines before and writing our own code there before executing it. The result of the code below to a random distribution of the objects produces a path like this:



Path that the robot will follow in the previous example

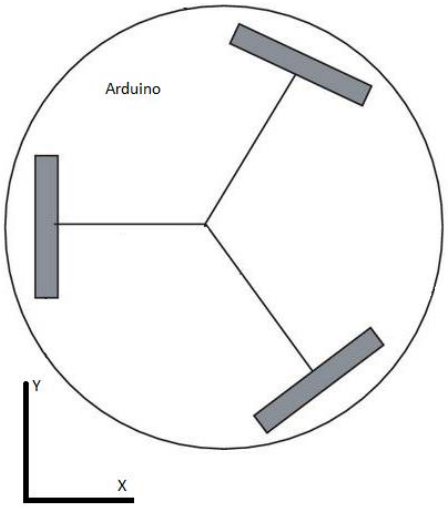
8.5 LOADING THE CODE TO THE ROBOT

Once we have the path that we want to follow, we have to load it to the Arduino of the robot. To begin we will open the Arduino program and open the file sketch_mov. This is a very basic program that has only one function in the loop which is scan. This function is defined later and is just a succession of orders to the serial.

We will open the file Arduino.txt which will be located at the project folder and make a copy-paste of all the text in this file to the function scan and then compile the Arduino code to the robot after connecting the computer and the robot by a wire.

Once it is done we just have to position the robot in the place where it should start and oriented as below. Turn both switches on and the robot will start the path.

It is very important to place the robot in the correct direction, to have the battery fully charged and to make sure that nothing will block it. When the battery is a bit discharged, we may find a wheel that receives a little less power than the others and with long tracks we can notice that the actual position is not the one where we should be.



Scheme of how the robot must be oriented.

9 CONCLUSIONS AND FUTURE LINES

At the beginning of this project I set myself three goals to achieve. Those ones were: being able to create a map with the objects in the scenario; being able to calculate the best route to get to anyplace in the scenario avoiding obstacles; being able to connect the robot with the computer for taking decisions and sending orders from the computer to the robot while the scene is running.

In my opinion, the first goal has been achieved successfully as it is possible to create a map with any distribution of the objects. The error found between the calculated position and the actual is not significant.

I also think, that the second goal has also been achieved as the code always finds a path to go from the robot to the desired position.

Finally, despite my intention, the third goal proposed in the introduction has not been possible to achieve due to, as explained previously, the connection between the Arduino board and the computer was impossible to do wireless.

However, after doing this project I think that the code created is quite good. Moreover, It can be used not only for this project as it can also be adapted to any other one that needs a mapping of a room from fixed cameras.

When doing the project I realized the potential and the magnitude it could have and for this reason I started to think in the possible improvements that could be added. If in the future I had more time and resources to work in this project, I would work in these lines.

When talking about the mapping part, the code could be improved and instead of looking for only one point in the map it could be possible to look for the area that every object fills.

Taking into consideration the route to follow, I would suggest to create a code that calculates a more natural path for the actor as the paths created with

the code now are made by straight lines and in only eight possible directions (each one at 45 degrees).

As the third goal was not achieved, I think that another future action could be linking both computer and robot because I think that it would be interesting to adapt the code to this situation and give the robot the possibility to interact with moving objects or other actors.

Another thing we could add in this code in the future would be a wider possibility of movements, as now the robot can only move from one place to another.

The development of this project has been a very interesting and entertaining way to combine into a single element, a multitude of disciplines viewed during the degree among which I must mention the control of the robot (this time for image processing) the use of artificial intelligence and in a lesser extent mechanics and electronics. It is splendid to see how things studied in the degree can match themselves like this and become something tangible that works correctly.

This work has also served to me to expand my knowledge into some areas that I had not worked much before and I am personally very interested in, which are the programming of a robot and its motion planning. So I am very glad of the opportunity I have had.

Finally, I would also like to thank Professor Andrea Bonarini for accepting me as his student and for all help, resources and advices I have been given during the entire project.